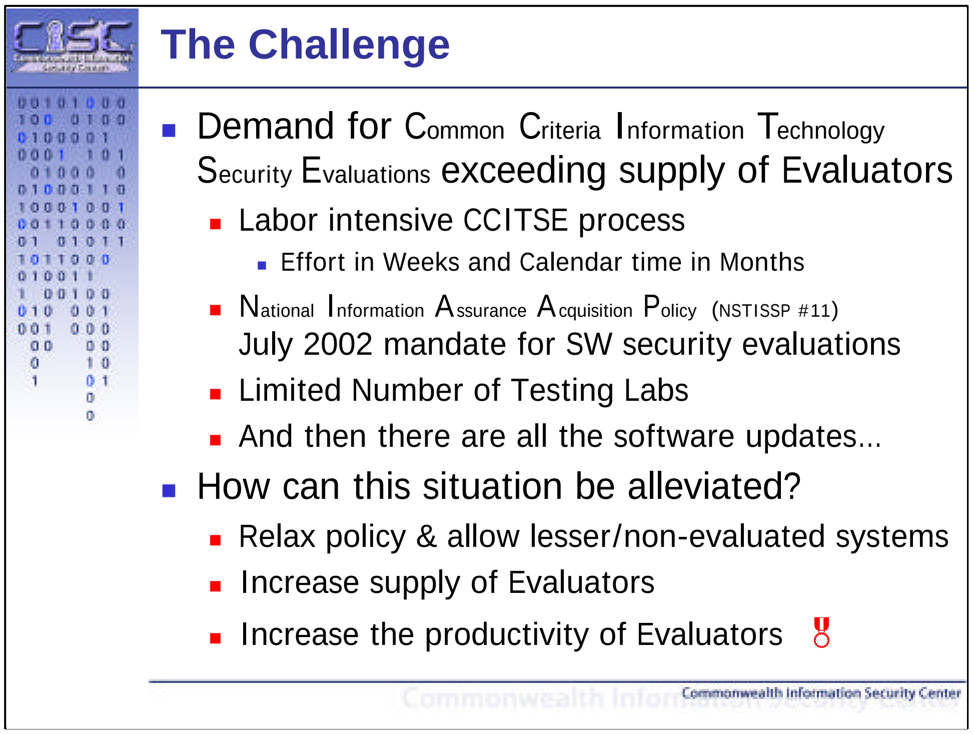
Software Security Impact Analysis

Principle Investigators: Shawn Bohner and Denis Gracanin

Ph.D. Students: Chang Dong, Boby George, Jianghui Ying, and Yunxian Zhou

Department of Computer Science

VirginiaTech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY



The Challenge

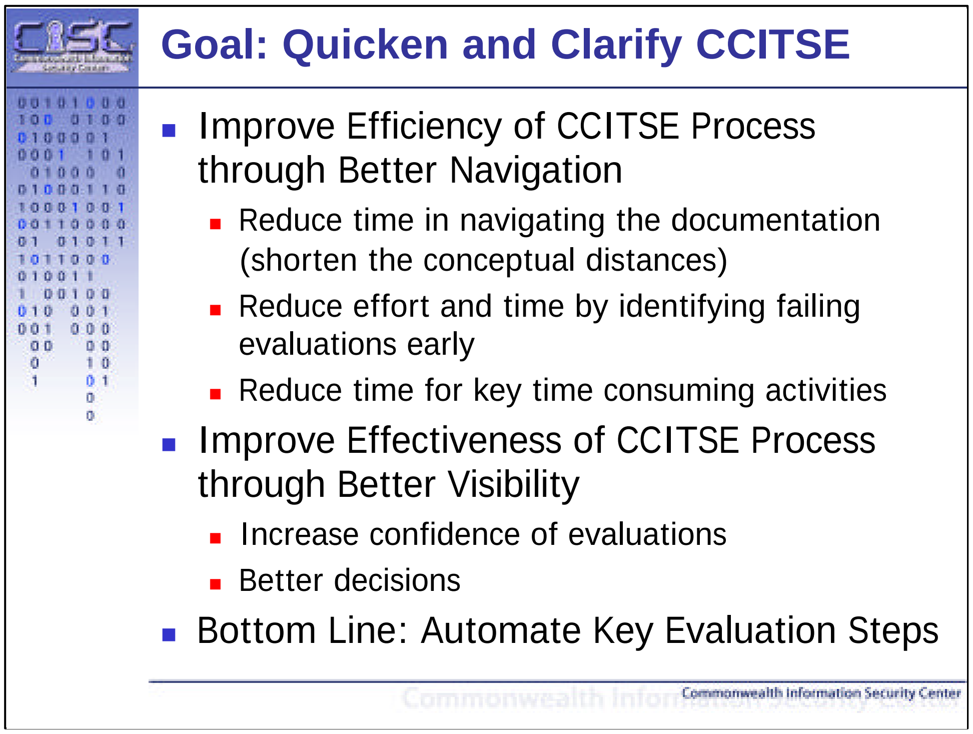
- Demand for Common Criteria Information Technology Security Evaluations exceeding supply of Evaluators
  - Labor intensive CCITSE process
    - Effort in Weeks and Calendar time in Months
  - National Information Assurance Acquisition Policy (NSTISSP #11) July 2002 mandate for SW security evaluations
  - Limited Number of Testing Labs
  - And then there are all the software updates...
- How can this situation be alleviated?
  - Relax policy & allow lesser/non-evaluated systems
  - Increase supply of Evaluators
  - Increase the productivity of Evaluators

Commonwealth Information Security Center

- Basic problem addressed by this research is one of a labor intensive CCITSE process coupled with instant demand brought on by NIAAP mandate for all IT security related software to be Common Criteria evaluated.
- This is exacerbated by the normal software evolution situation where software updates (versions…) must also be re-evaluated.
- Unfortunately, the demand exceeds the supply and evaluations are predictably becoming the bottleneck for government security software acquisition.

There are three obvious responses to the situation

1. Relax policy – a non-starter
2. Full-employment act for evaluators – like Y2K, mobilizing this number of resources will be costly
3. Automate key areas of the CCITSE process that consume the largest portions of time and effort. – Our solution fits here ;-)



Goal: Quicken and Clarify CCITSE

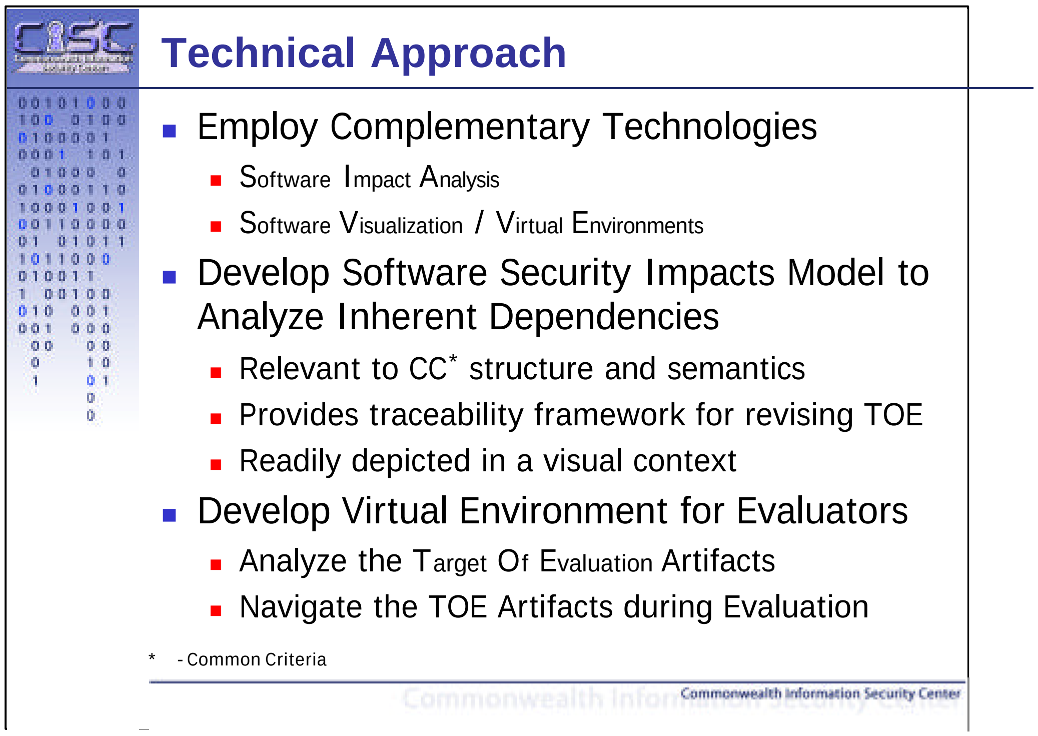
- Improve Efficiency of CCITSE Process through Better Navigation
  - Reduce time in navigating the documentation (shorten the conceptual distances)
  - Reduce effort and time by identifying failing evaluations early
  - Reduce time for key time consuming activities
- Improve Effectiveness of CCITSE Process through Better Visibility
  - Increase confidence of evaluations
  - Better decisions
- Bottom Line: Automate Key Evaluation Steps

Commonwealth Information Security Center

From a business perspective, what are the goals of this research?

1. Improve efficiency by automating navigation through the enormous amounts of software documentation and source code during the evaluation process. – You will probably get a comment here about most systems under evaluation come with little documentation. The appropriate reply is that these are the ones that are failing the evaluation process. The vendors are paying big bucks for the TOE to be revised (not to mention the delayed time before someone starts buying their product ;-)

   Note that the map created for navigation has a great feature of providing an initial analysis if the evaluation will fail – that is, if the map is not complete enough for navigation, it is probably an indication of likely failure and a pruning opportunity for an overburdened evaluation process.
2. Improve effectiveness with better visibility – the impact analysis provides a significant structure from which to determine if a software product meets the Common Criteria and even a mechanism for determining how well the system meets them. This increases confidence of the evaluators (early thus saving time) and results in better decisions that thwart vulnerabilities.



Technical Approach

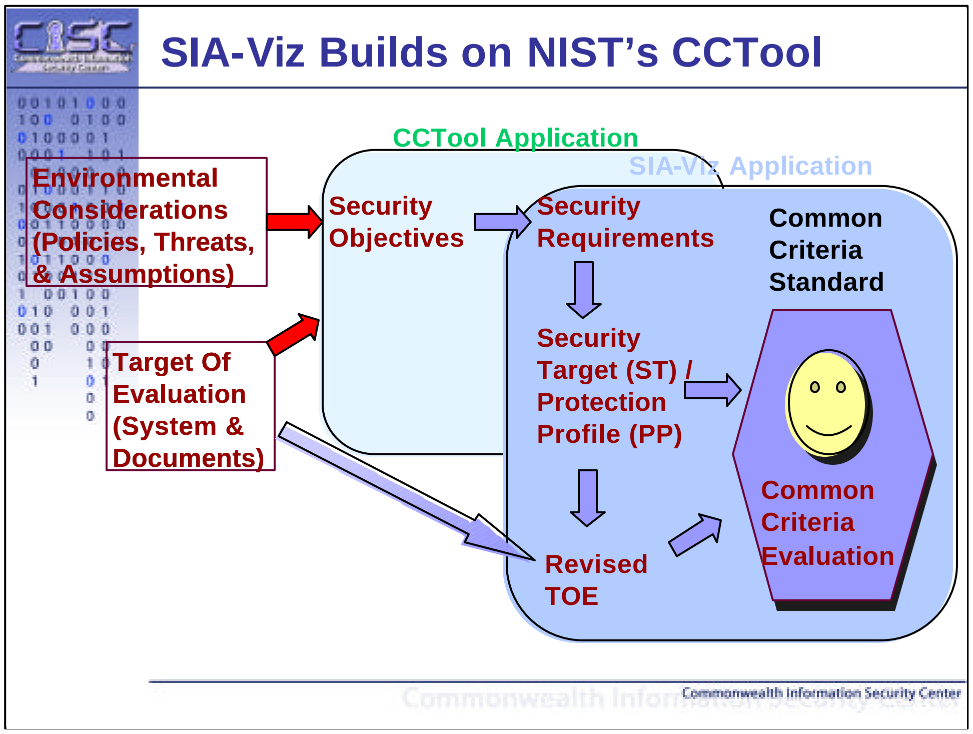
- Employ Complementary Technologies
  - Software Impact Analysis
  - Software Visualization / Virtual Environments
- Develop Software Security Impacts Model to Analyze Inherent Dependencies
  - Relevant to CC* structure and semantics
  - Provides traceability framework for revising TOE
  - Readily depicted in a visual context
- Develop Virtual Environment for Evaluators
  - Analyze the Target Of Evaluation Artifacts
  - Navigate the TOE Artifacts during Evaluation

* - Common Criteria

Commonwealth Information Security Center

Our technical approach is to combine the benefits of two proven technologies that in this context complement each other.

1. The first is software (change) impact analysis. This technology uses the dependency (formal and non-formal) relationships between software life cycle objects (SLOs) to identify the potential ripple-effects of software changes (also used to identify software architecture structures). This dependency representation form, augmented with security dependencies, provides a good basis to identify elements of interest for CC evaluators. However, in a textual or 2 dimensional form, these analyses are quite complex.
2. The second technology, Virtual Environments, uses 3 dimensional visualization coupled with metaphors that allow "immersion" in the representation – thus allowing better visualization of large corpuses of information and better navigation to constituent parts. In particular, this applies to all of the elements of the TOE as the CC evaluator engages in examination.
3. Bottom line here is that the SIA and VE technologies work synergistically with evaluators to alleviate many of the bottlenecks in the CCITSE process.



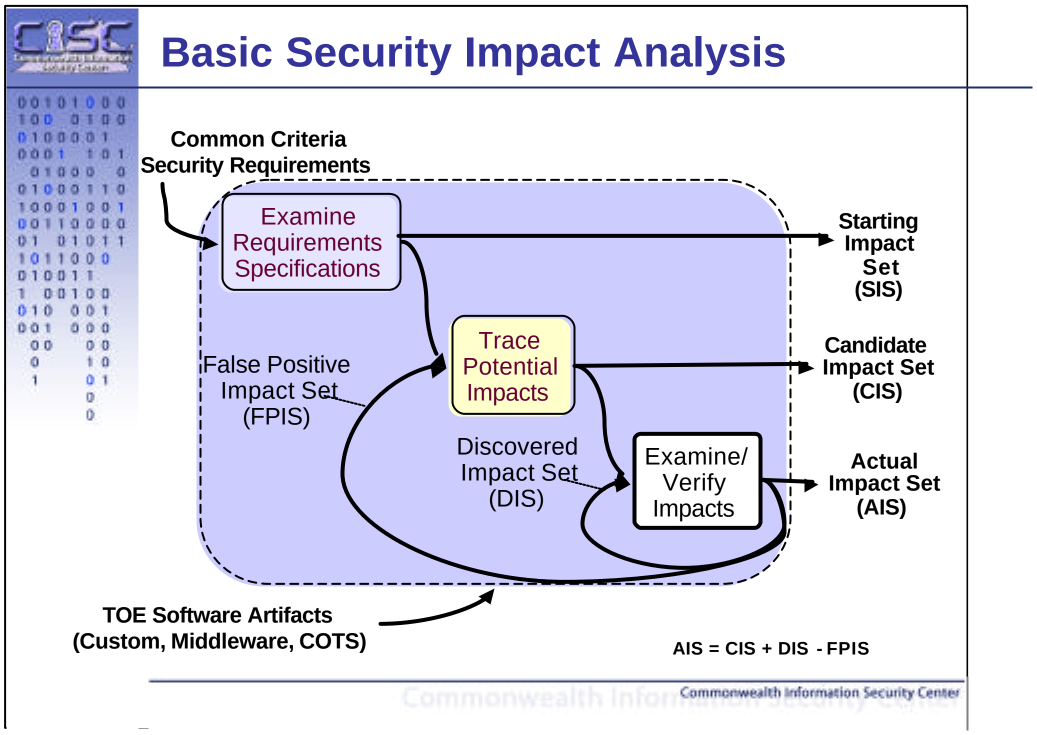
SIA-Viz Builds on NIST's CCTool

CCTool Application
SIA-Viz Application
Environmental Considerations (Policies, Threats, & Assumptions)
Target Of Evaluation (System & Documents)
Security Objectives
Security Requirements
Common Criteria Standard
Security Target (ST) / Protection Profile (PP)
Revised TOE
Common Criteria Evaluation
Commonwealth Information Security Center

This slide shows how the SIA-Viz environment augments the CC tool (a recognized tool in the NIST/NIAP suite). The CCTool is an instrument to help with the production of the Security Target (report) and Protection Profile (report). For the most part, these are similar reports that are a product of interviews with subject matter experts.

This slide has a series of builds starting with the overall information flow into the CC evaluation. The second build shows the application of the CCTool focusing on Security Objectives, Security Requirements, and the ST/PP report.

The final build shows where SIA-Viz builds on the CCTool by introducing key dependencies for the Revised TOE and a mechanism to support the CC Evaluator in the analysis and navigation aspects of evaluation.



Basic Security Impact Analysis

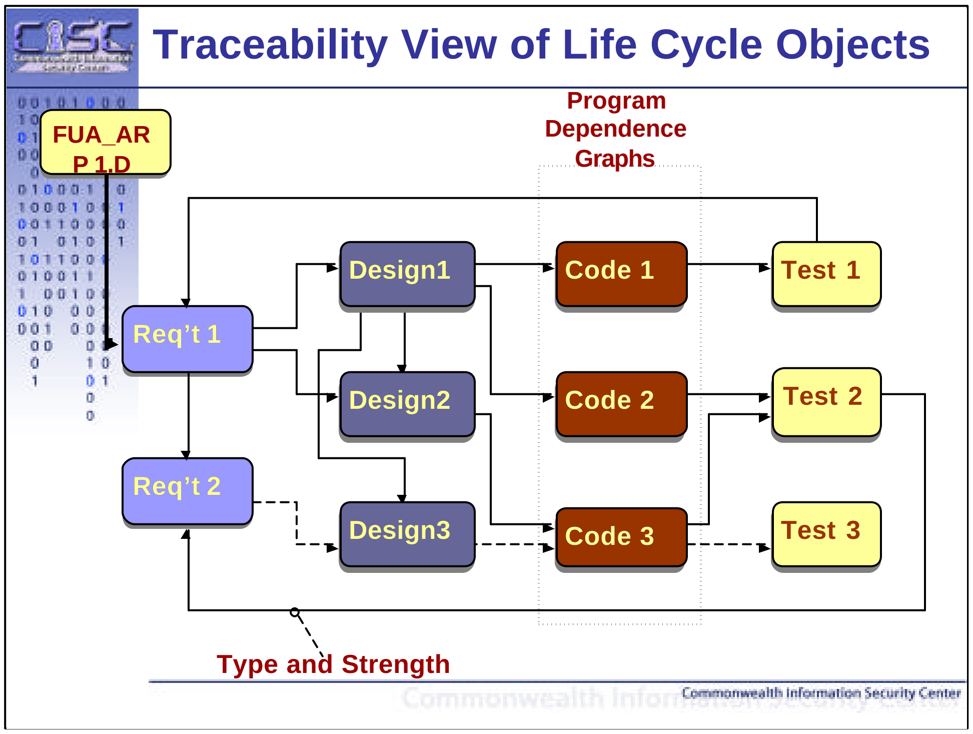
Common Criteria Security Requirements
Examine Requirements Specifications
Starting Impact Set (SIS)
Trace Potential Impacts
Candidate Impact Set (CIS)
False Positive Impact Set (FPIS)
Discovered Impact Set (DIS)
Examine/ Verify Impacts
Actual Impact Set (AIS)
TOE Software Artifacts (Custom, Middleware, COTS)
AIS = CIS + DIS - FPIS
Commonwealth Information Security Center

Impact analysis is an iterative process, starting with some seed information, and systematically identifying impacts (dependency-based) to result in some findings. For software security, with all of its informal information, it is often necessarily a human in the loop activity – seeking guidance and course correction.

Essentially, after the TOE has been revised with the dependency information in place, the SIA process proceeds as follows:

1. A set of CC related security requirements are identified.
2. The direct and indirect impacts are traced (through the automated facilities) to outline the potential impacts.
3. The potential impacts life cycle objects are examined and verified for the evaluation – discovered impacts (those not identified automatically) are added to the known impacts and those deemed not relevant (false positives) are eliminated from the potential impacts.
4. Iterate on the above three steps until there is a convergence on the actual impact set.

This process is largely automated and produces a map for the evaluator to use in the evaluation. The map is often complex and contains considerable information that is not readily represented in a 2D form.



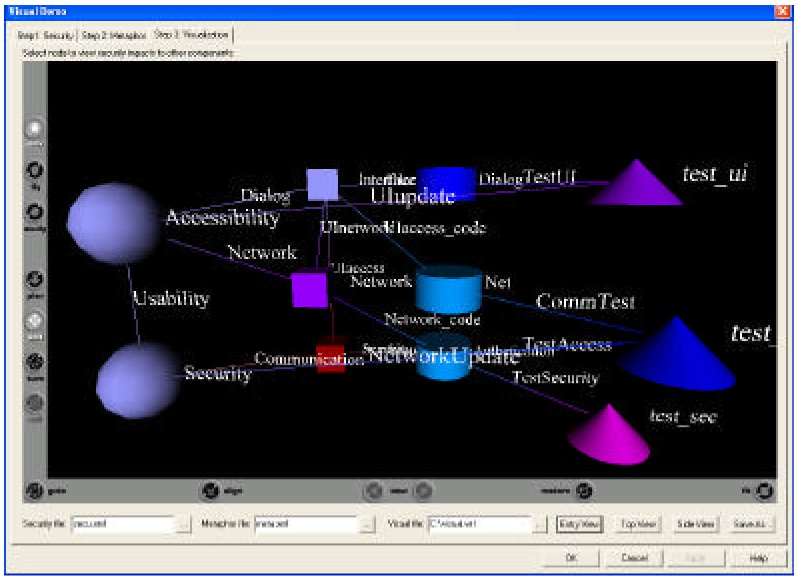
Traceability View of Life Cycle Objects

The simplified analysis described on the previous slide provides the basis for analyzing and navigating the manifold dependencies in the TOE. This slide takes that process the next step showing how the tracing is mapped among the life cycle objects.

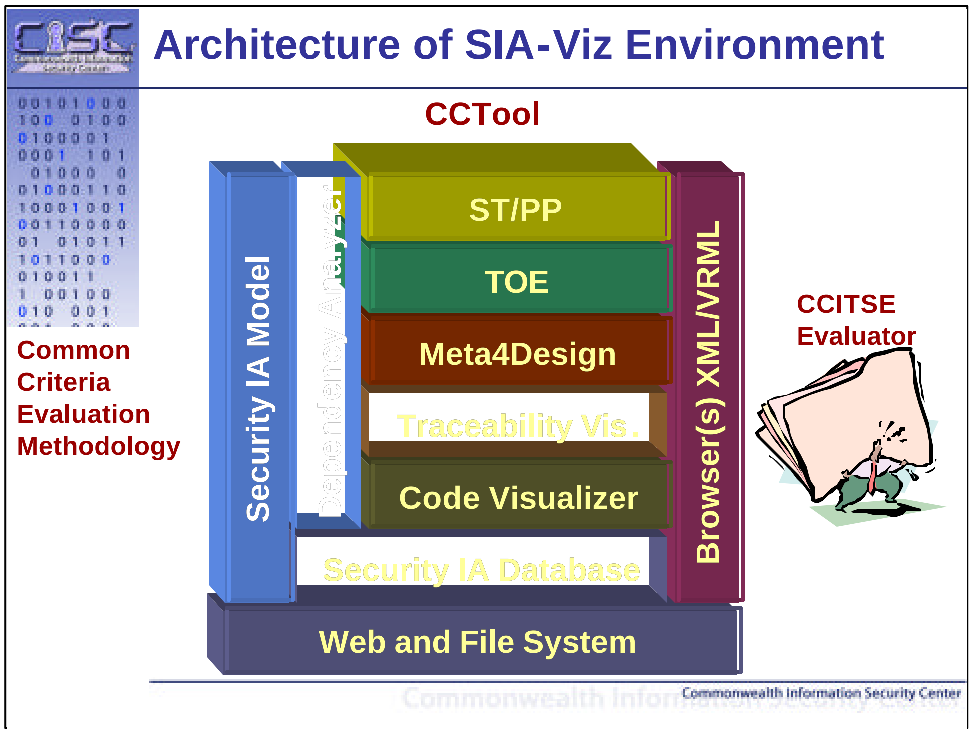
There are a series of builds that go with the dialogue as follows:

1. The FUA_ARP 1.D component on the left traces directly to "Req't 1" and subsequently to "Req't 2."
2. The next build introduces traces to the Design components, a less abstract artifact of the development. Note that the dotted line depicts the idea of an indirect dependency and that subsequent dependency relationships are indirect.
3. The next build introduces traces to Code components.
4. The next build then closes the loop with test components. Actually, dependencies for test are related back to their respective life cycle objects – e.g., an integration test would also have a Design component dependency relationship. We simplified this slide to convey the traceability relationships as they pertain to impact analysis.
5. The next build shows the Program Dependence Graphs as the formalism used with source code – software code analysis has a long record of analyzing dependencies and we stand on broad shoulders here.
6. The next build introduces the notion that "not all dependencies are created equal." While we use general graph theoretic techniques for

Visualizing and Navigating the Impacts



Security file: secu.xml
Metaphor file: meta.xml
Visual file: C:\visual.wrl



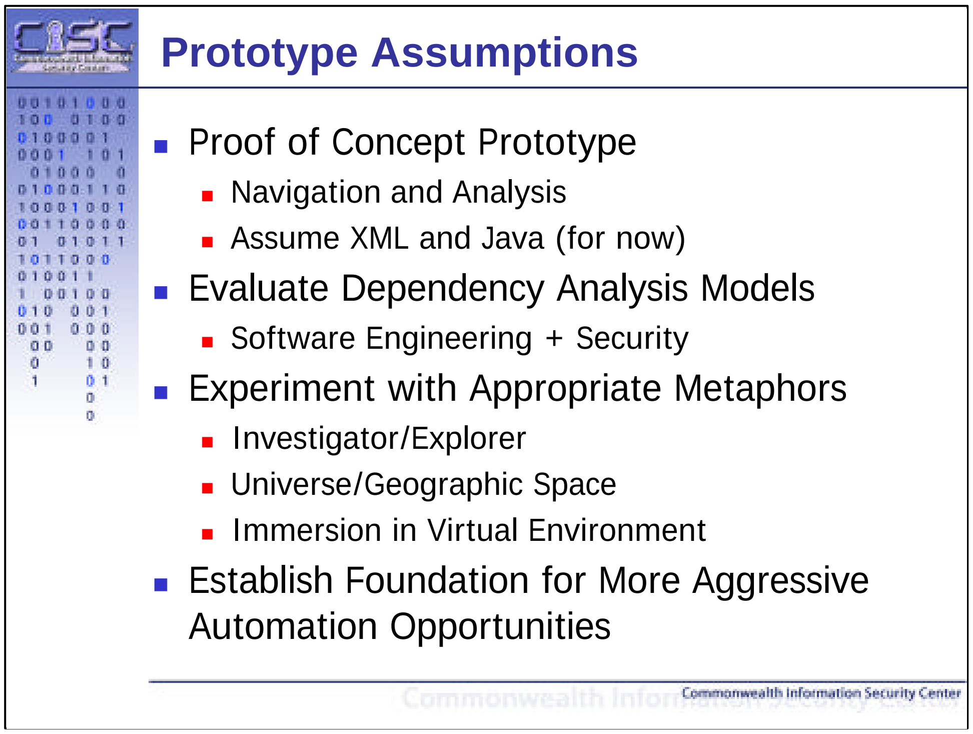
Architecture of SIA-Viz Environment

This slide outlines the basic architecture of the SIA-Viz Environment. We exploit the fact that the software artifacts in the TOE are available on the computer file system and the web. We can use HTML/XML addressing to navigate the documents readily. From the Common Criteria Evaluation Methodology, we developed a Security Impact Analysis Model. This provides the formal representation that connects SLOs through a network of dependencies. From this, we are developing the dependency analyzer to "slice" out the relevant elements for the Evaluator to examine. This information is feed into the Security Impact Analysis Database for further slicing and analysis during the navigation, and feed into the Traceability and source Code Visualizers to prepare it for investigation using the XML/VRML-based Browsers.

While the Traceability and Code Visualizers are separate at this time, we plan to integrate them. We've discovered that source code will not always be the only formal representation we can glean dependencies from automatically. Increasingly, products like Rational provide intermediate forms of requirements models and design that can be mined for dependencies.

Also, with our aim to develop a flexible metaphor mechanism that can evolve with the evaluator's needs, we have built the META4Design tool to support the visualization aspects of the artifacts.

Note that at the top of the architecture is the ST/PP (produced by



Prototype Assumptions

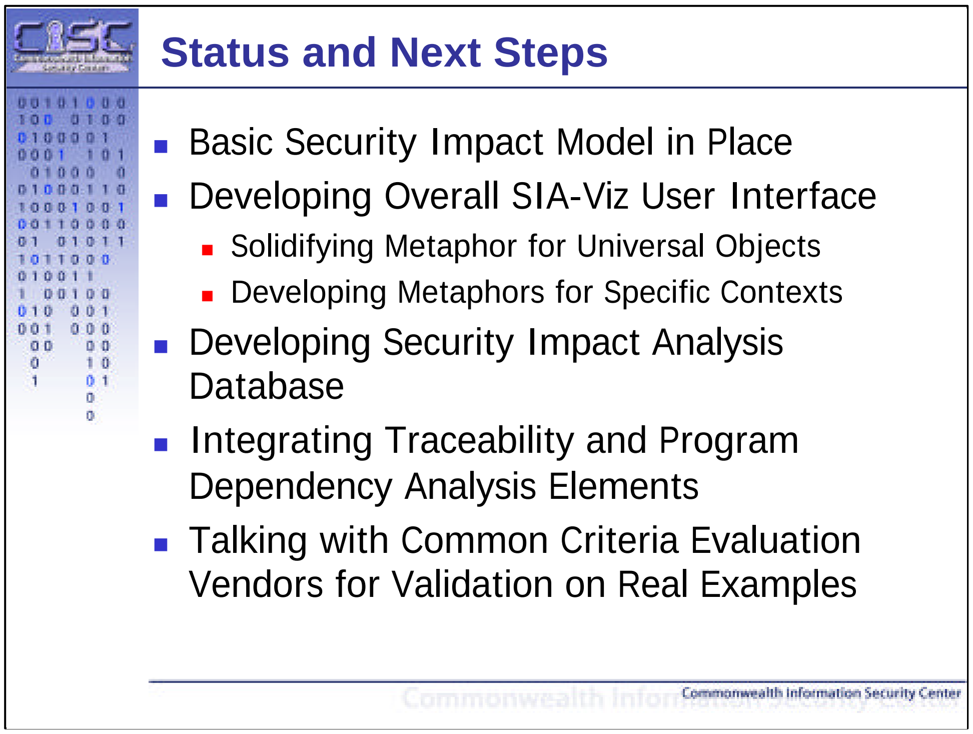
- Proof of Concept Prototype
  - Navigation and Analysis
  - Assume XML and Java (for now)
- Evaluate Dependency Analysis Models
  - Software Engineering + Security
- Experiment with Appropriate Metaphors
  - Investigator/Explorer
  - Universe/Geographic Space
  - Immersion in Virtual Environment
- Establish Foundation for More Aggressive Automation Opportunities

Commonwealth Information Security Center

It is important to clarify expectations for this work. It has only started 6 months ago, therefore we have scoped the proof of concept prototype to focus on supporting the Evaluator's navigation and analysis activities. We have started with the future in mind by using XML and Java as our demonstration situation – while C/C++ and specialized documentation environments dominate today, we believe that XML and Java will emerge as representations of choice in future security related systems development.

Our short-term aims are to evaluate and refine the Dependency Analysis Models integrated from software engineering and security methods.

Also, we must experiment with the appropriate metaphors for the CC Evaluators. To this end, we are currently examining Investigator/Explorer perspectives of the CC Evaluator. We want to have the virtual environment allow the Evaluator's perspective to change as he/she navigates the corpus of SLOs. When he/she examines a requirement, the environment changes to that requirement's view of the universe – only seeing things that pertain to the requirement; important things close, others far. This is why we are now looking at a Universe/Geographic Space for our metaphor. These elements will allow us to immerse the Evaluator in the Virtual Environment to carryout his/her tasks.



Status and Next Steps

- Basic Security Impact Model in Place
- Developing Overall SIA-Viz User Interface
  - Solidifying Metaphor for Universal Objects
  - Developing Metaphors for Specific Contexts
- Developing Security Impact Analysis Database
- Integrating Traceability and Program Dependency Analysis Elements
- Talking with Common Criteria Evaluation Vendors for Validation on Real Examples

Commonwealth Information Security Center

Future Targets include:

•Conduct Experiments to Demonstrate Benefits of the Automation

•Refine our Architecture

•Extend Automation Vision to Support Revision of TOE

•Templates already loaded with dependencies

•Refine Dependency Analysis Engine for Better Accuracy

•Incorporate Semantic Design's DMS

•Incorporate Emerging Standards like X3D to Improve Collaborative VE